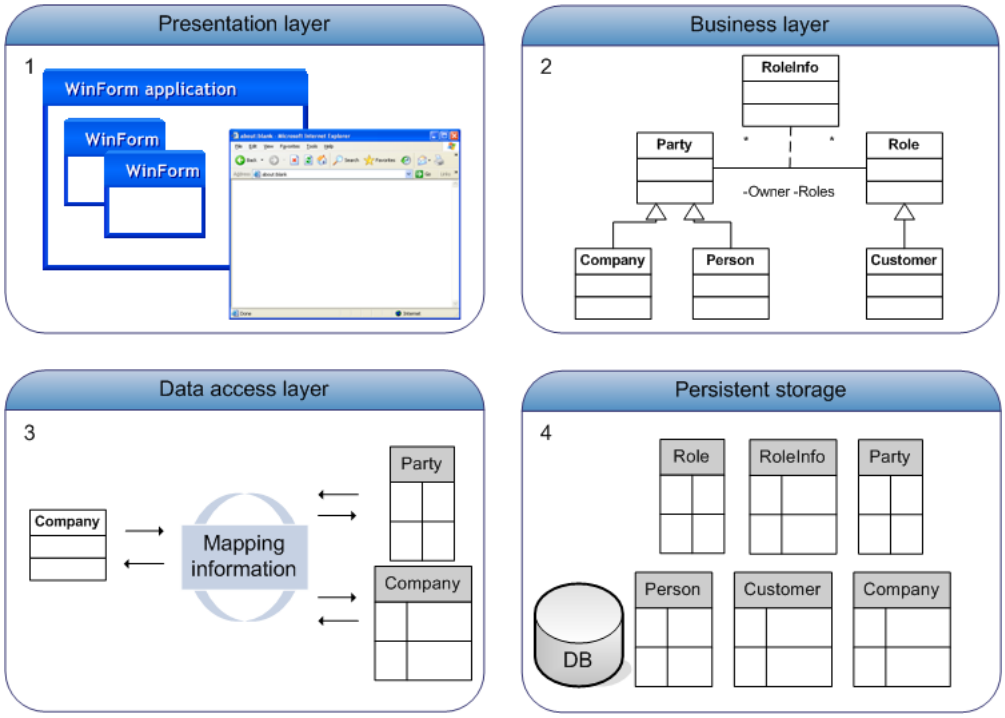
Why ECO?

Table of Contents

1 N-Tier application development

N-Tier application development is the technique of separating developed applications into logically separate layers. These layers may be on separate computers or the same computer, the importance of this approach is the logical separation whereas physical separation is a technique made available through the implementation of logical separation.



The presentation layer

The purpose of this layer is to present information to a user, and to record input from the user or receive other instructions such as updating the data storage, navigating to other data, executing reports, etc.

The business layer

The purpose of this layer is to reflect as accurately as possible the entities of the business the application is being designed to emulate (the "Business domain"). These class entities will not only contain properties for holding information about the business objects but will also contain methods which may be executed in order to reflect the behavior of these entities too. Classes will typically have names like "Customer", "Employee", and "Order", however these will change depending on the business domain being implemented.

The data access layer

In real life a business entity such as a Customer has no ability to save or retrieve its information from a data storage, it makes sense therefore that classes within the business layer do not have such features. The purpose of the data access layer is to provide the application with a way of making the lifetime of a class instance extend beyond the point where the application is closed. This layer will retrieve data from the specified data storage (which may or may not be a database) and transform this data into a instance of the relevant class from the business layer, in addition this layer is responsible for

updating the data storage with changes made to business class instances, and also to delete data from the persistence storage where business class instances have been permanently destroyed. Not only does this layer make persistence transparent to the business classes layer, it also makes it possible to swap out one data storage and replace it with another without having to rewrite any of the application source or the business classes' source.

Persistent storage

The persistent storage is responsible for persisting, retrieving, and deleting permanently stored data as instructed by the data access layer. This persistent data is then ultimately available to the application across multiple application restarts as if the business class instances within the application had lived beyond the lifetime of the application.

This clear separation provides the following advantages:

1. It is easier to switch between different persistence stores or to determine which type of data storage to use at runtime according to the user's application settings.
2. Developing separate layers helps to prevent business logic from creeping into the presentation layer where it is hard to find.
3. Writing business logic and validation into a separate layer allows multiple areas of the presentation layer to adhere to the same business rules, and makes it less work to implement a secondary presentation layer (VCL.NET / Web).

2 ECO features

ECO promotes the N-Tier approach to ensure that your applications are well structured and easier to maintain, however, in ECO there is no need to develop the data access layer as ECO possesses this ability internally. ECO has numerous persistence abilities, including:

- Load / save to a local XML file.
- Load / save to numerous SQL databases as standard (SQL Server, InterBase, Firebird, MySQL, DB2, and more).
- Automatically generate the DB schema of a SQL database from the model.
- Automatically upgrade (evolve) the DB schema based on changes to the model whilst preventing data loss.
- Load / save objects across multiple persistence stores, including the ability to save to mixed DB servers or to save using distributed transactions (SQL Server).
- Map objects to / from an existing database structure by providing XML mapping information.
- Use a remote ECO persistence server so that no DB server components or database credentials are required on the client, the server may switch to a new persistence storage without having to update the clients.
- Reverse engineer an existing database in order to create a business model.

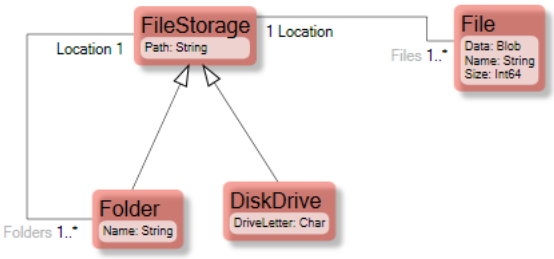
The aforementioned features are usually provided by a custom data access layer written by the developer, usually the data access layer written does not provide all of these features and may take a considerable amount of time to develop. ECO's persistence abilities will inevitably save many hours of development, in fact a simple business model can be created, the code generated, and a database created to persist the business objects in less than five minutes.

An application development framework

ECO provides the developer with much more than a way to easily persist business objects within a persistence storage. In addition to the data access layer ECO consists of a number of service modules which combine to create an impressive set of tools which aid the developer in creating robust applications in a much shorter timeframe. These include (but are not limited to):

2.1 Automatic source code generation

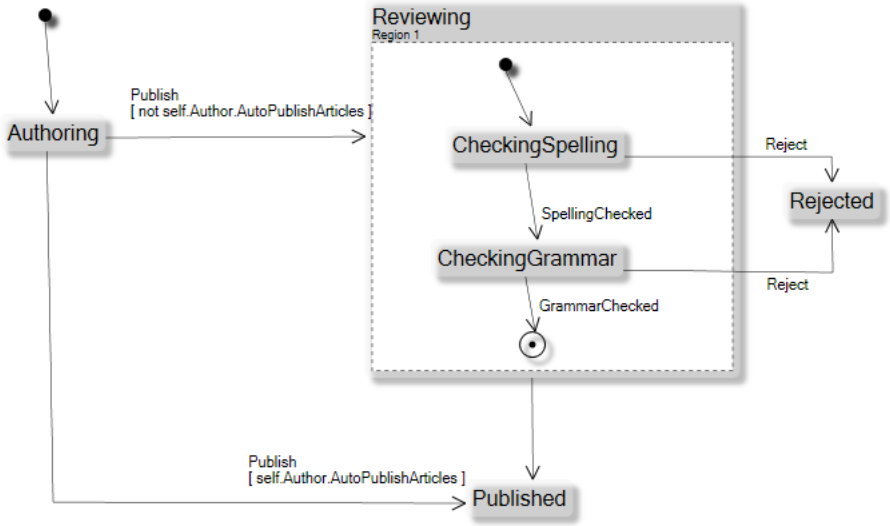
ECO is akin to a UML model executor. Your business model is first defined using a mixture of UML classes and state diagrams, source code for your business classes is generated or updated automatically from this model. The developer may then modify this source code in order to implement modeled methods, add new methods, etc. Once source code has been generated it is still perfectly safe to refactor your business model by adding classes, renaming classes, renaming methods and so on; the ECO source generator is a "merge generator" so your changes will not be lost.



2.2 UML state diagram execution

Each business class in your model may optionally have an associated UML state diagram. Once defined and source code has been updated ECO is capable of executing these state machines. Features include:

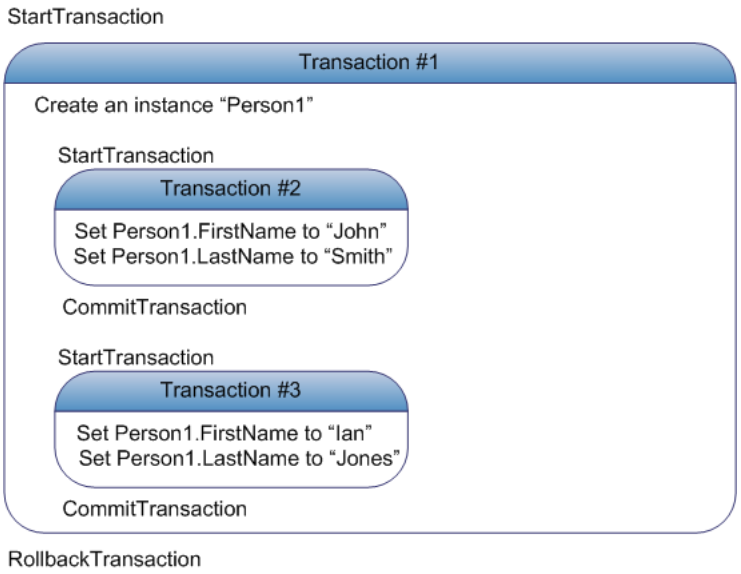
- Triggers: Signals to ECO that a transition should be made from the current state to a new state.
- Guards: An OCL (Object Constraint Language) expression that indicates whether a specific transition is currently permitted or not. When multiple transitions use the same trigger the guard expression determines which transition to take.
- Sub regions: Each state may have zero or multiple concurrent regions, allowing your state diagrams to be as simple or complex as required.



2.3 In-memory transactions

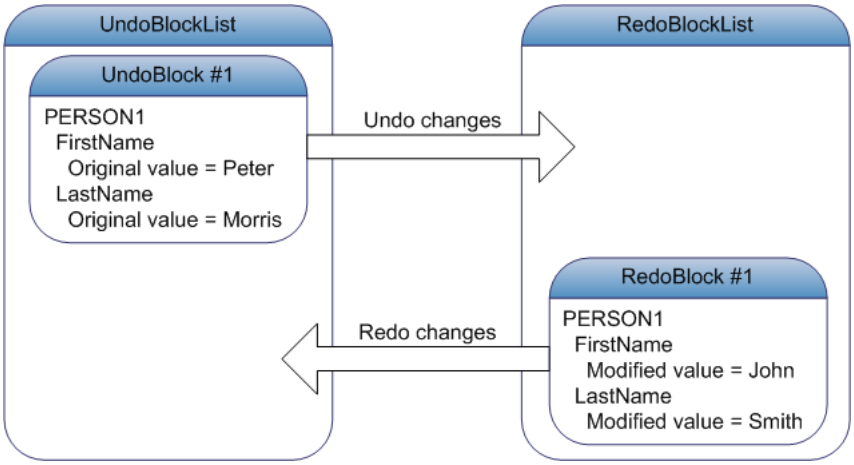
All changes to your business objects are automatically detected by ECO. By using in-memory transactions it is possible to rollback or commit groups of changes as a single unit of work. Modifications to object properties, object creation and even object deletion may be reversed by rolling back an in-memory transaction, resetting the entire state of the "EcoSpace" back

to the exact state when the transaction was started.



2.4 Multiple undo/redo block

As with the transaction feature all changes to the business objects are automatically detected and recorded into an "undo block", these changes can then be reversed and reapplied numerous times. This allows the developer to operate multiple independent operations which may be reversed or reapplied in any order they wish, one usage example of this feature would be to enable the developer to have multiple Forms open at the same time and to allow the user to undo or redo the changes in each one independently.



In the example above an undo block was created.

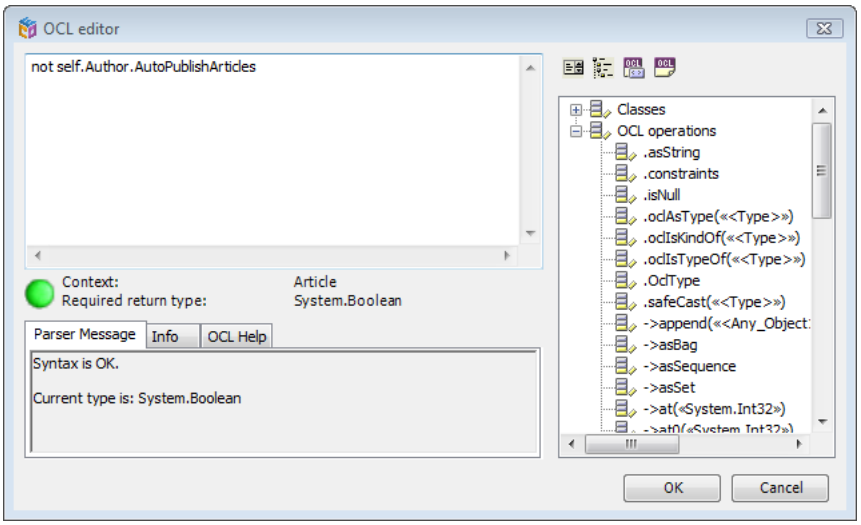
1. When the user changed Person1 FirstName/LastName to John/Smith ECO recorded the original values Peter/Morris in the undo block.

2. The changes of the undo block were then reversed.

3. The undo block was removed from the UndoBlockList and placed into the RedoBlockList.
4. Now instead of recording the original values the modified values John/Smith are recorded.
5. The user is able to re-do and undo their changes multiple times.

2.5 Expression evaluation

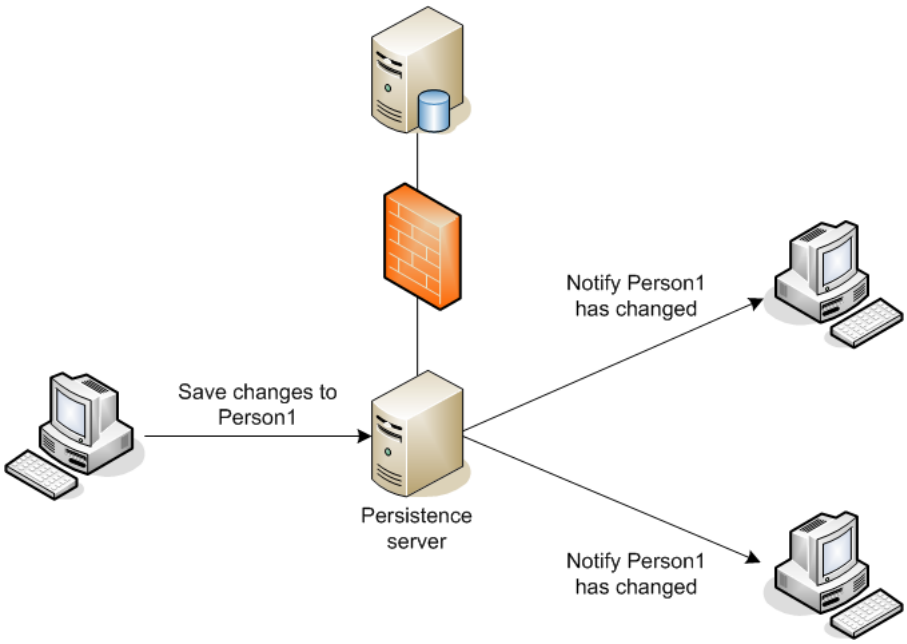
ECO implements the Object Constraint Language "OCL" as specified by The Object Management Group. This allows the application to easily make calculations based on business objects without having to write any code. For example the number of published articles an author has could be evaluated as "self.Articles->select(State = #Published)->Size", where the state is evaluated from the Article class's state machine diagram illustrated earlier. All OCL input is aided by an OCL expression editor, this editor provides the developer with a context sensitive guide to all currently available OCL operations, and an indication as to whether or not the entered expression is valid plus the return type of the expression.



The OCL is also used by the various ECO components for retrieving business class instances from the persistent storage and identifying data to databind to.

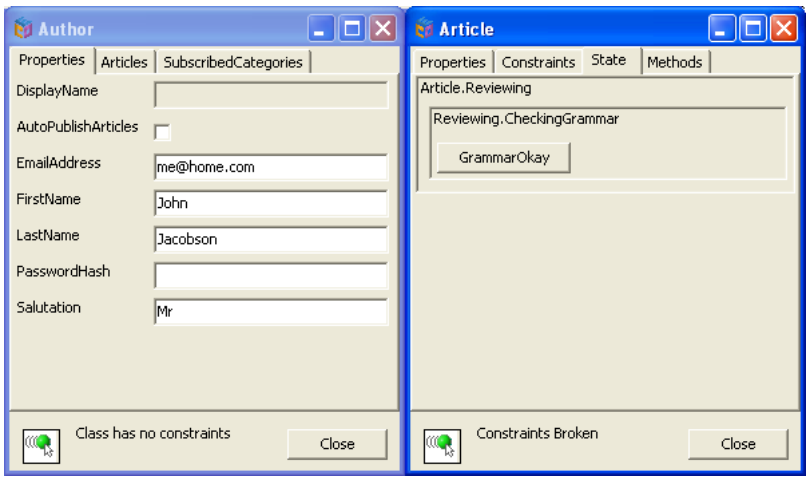
2.6 Object cache synchronization

ECO allows you, if you choose, to defer all persistence operations to a custom persistence application on a shared server rather than connecting to the persistent storage directly. This central place for serving persistence requests allows the server application to automatically track which business objects have been modified since a given time, it can therefore notify each client whenever data it is using has been updated by another user, saving the client application from having to re-fetch data repeatedly in order to guarantee that the client is always looking at current data.



2.7 Quick application prototyping

ECO provides the ability to test run the business model during design time. By generating forms for object instances automatically from model information it is possible to manipulate instances of your business classes without writing a single line of code. These forms include properties, associations to other business objects (including full drag and drop support) and state machine support. These enable you to quickly test and revise the business model you have designed by entering data into it without having to write a single line of code. Modeling mistakes in the business model may often be identified quickly as soon as you attempt to enter data into your business objects, eliminating the need to manually create your forms to enter this data will improve the speed of prototyping your application dramatically.



2.8 Other features

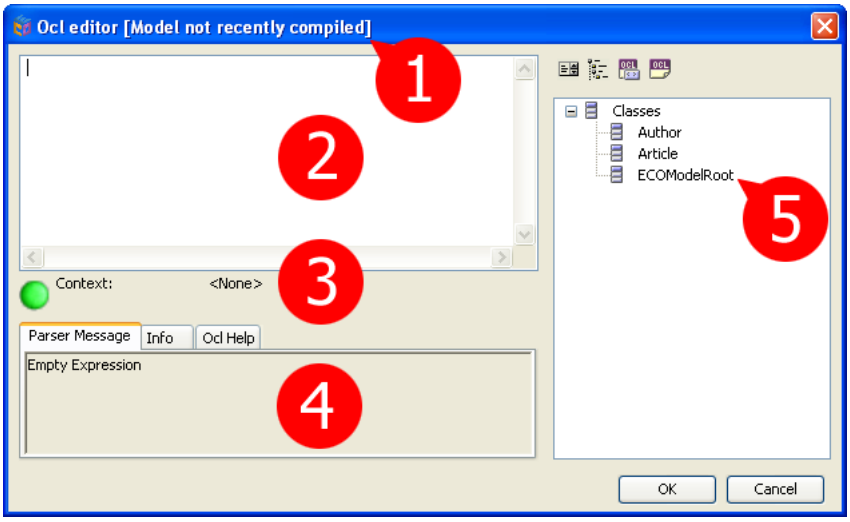
Other features include:

- Object contents caching. Data is requested from the persistence storage on demand simply by referring to objects that have not yet been loaded (via code such as Company.Offices, or via DataBinding), once loaded the contents of fetched objects are cached locally and then only re-fetched from the persistent storage if the data in the cache is marked "Invalid" either by the developer or as a of a remote persistence synchronization.
- Object properties that have contents requiring lots of memory / network bandwidth (for example, documents or images) may be marked "Delayed fetch". The contents of these properties and not fetched with the rest of the object's properties but are instead only fetched when the application first tries to read them.
- Automatic caching and invalidation of calculated properties. Using the object constraint language it is possible to create non-persistent calculated (derived) properties. The value of these properties are calculated the first time it is requested and then cached, the cache is automatically invalidated if any of the elements required to derive the result alter (other classes, other properties, etc). This approach can save a lot of CPU work, especially if the calculation involved is intensive.
- Object versioning. When a class in the model is marked as "Versioned" changes to instances of the class are historically recorded, it is then possible to navigate through objects / associations etc as they were at a given date and time.
- Access to full model information at runtime.

As you can see ECO goes far beyond the N-Tier approach, the advanced object relational mapping support it provides is merely a small part of the entire framework. Hopefully this series of quick start documents will illustrate how quickly ECO applications can be prototyped and developed. By the end of this series you should be familiar enough with the basics of the ECO application framework to be able to create your own business applications, we hope that your interest will be suitably aroused and will wish to learn more.

3 Using the OCL editor

ECO provides an OCL editor form which aids the developer in entering valid OCL expressions. Whether invoked during the modeling of business classes or when specifying expressions on an ECO handle, this editor will validate the OCL expression against the model and also provide a list of valid OCL operations. Before starting these quick start tutorials you may wish to familiarize yourself with this form. Apart from providing a context sensitive way of defining OCL expressions it is also a useful tool for validating expressions have been entered correctly. This form is invoked in various places throughout the IDE as an editor whenever an OCL expression is required.



The preceding screenshot shows the OCL editor:

1. The caption of the form informs us that the model has not recently been compiled, compiling the application will ensure that the binaries are up to date and therefore the model information presented in the OCL editor form will contain the most recent information.
2. The large text area directly beneath the caption is where the developer enters the OCL expression.
3. The context for the expression is displayed just beneath the input area. If the context is "Author" then valid expressions will additionally include "self.FirstName" etc.
4. As the expression is altered it is validated immediately. The result of the evaluation is displayed in the read-only area at the bottom of the form.
5. To the right of the form there is a treeview displaying a list of OCL operations you may wish to add to the end of the current expression. As the result type (#4) changes this treeview's contents will also alter. For example, if the result of the evaluation is "Author" then a "Roles" node will appear containing a child node entitled "Articles"; whereas if the result of the evaluation is "String" then an "OCL operations" node will appear containing child nodes such as "Length", "Substring", and "IndexOf". Double-clicking a node in this list will append the selected OCL operation to the current expression.

4 Summary

As you can see ECO goes far beyond the N-Tier approach, the advanced object relational mapping support it provides is merely a small part of the entire framework. Hopefully this series of quick start documents will illustrate how quickly ECO applications can be prototyped and developed. By the end of this series you should be familiar enough with the basics of the ECO application framework to be able to create simple business applications, we hope that you will have been bitten by the ECO bug and will wish to learn more.

As more documents are added to this series they will be published on our website at http://www.capableobjects.com.

Index